Web Service Composition with User Preferences

Naiwen Lin

Department of Computer Science,
University of Maryland, College Park, MD 20742, USA,
nwlin@cs.umd.edu

Abstract. In Web Service Composition (WSC) problems, the composition process generates a composition (i.e., a plan) of atomic services, whose execution achieves some objectives on the Web. Existing research on Web service composition generally assumed that these objectives are absolute; i.e., the service-composition algorithms must achieve all of them in order to generate successful outcomes; otherwise, the composition process fails altogether. The most straightforward example is the use of OWL-S process models that specifically tell a composition algorithm how to achieve a functionality on the Web. However, in many WSC problems, it is also desirable to achieve users' preferences that are not absolute objectives; instead, a solution composition generated by a WSC algorithm must satisfy those preferences as much as possible. In this paper, we first describe a way to augment OWL-S process models by qualitative user preferences. We achieve this by mapping a given set of process models and preferences into a planning language for representing Hierarchical Task Networks (HTNs). We then present SCUP, our new WSC algorithm that performs a best-first search over the possible HTN-style task decompositions, by heuristically scoring those decompositions based on ontological reasoning over the input preferences. Finally, we discuss our experimental results on the SCUP algorithm.

1 Introduction

Web Service Composition (WSC), dynamic integration of multiple Web Services to fulfill the requirements of the task at hand, is one of the most important area in the Web Services research. Most research on composing Web services has been focused on developing new algorithms based on existing AI planning techniques. Examples of this approach include [1–6]. All of these works generally assumed that the composition process generates a solution composition (i.e., a plan) of atomic services, whose execution achieves some objectives on the Web. Those objectives are almost always required to be *absolute*; i.e., a service-composition algorithm must achieve all of them in order to generate successful outcomes; otherwise, the composition process fails altogether. As an example, in WSC planning systems such as [2, 4, 6], such absolute objectives and constraints specify structural ways of decomposing higher-level process models in OWL-S [7] into an atomic-level composition that solves the input WSC problem.

In many real-life WSC scenarios, on the other hand, the aim is to generate a solution that not only achieves some absolute goals and any constraints associated with them,

but also is desirable with respect to some user-provided preferences. For example, there are abundant number of travel Web sites providing transportation and accommodation services. Possibly, there will be several combinations of these services that will allow one to plan for a trip from a source location to a destination. However, an agent making such trip arrangements will need to take into account user preferences such as a desired range on the total cost of the trip, preferences on particular transportation companies/hotels, and certain times/dates for the trip. None of these are necessarily absolute goals in the sense that, if violated, they do not affect the correctness of the compositions but specify that certain compositions are more preferable than others.

In this paper, our contributions are as follows:

- We describe a way to augment Semantic Web services described as OWL-S process models with user preferences. We first describe a planning formalism that is based on the previous languages developed for *Hierarchical Task Networks (HTNs)*. Then, we describe a way to take OWL-S service descriptions and preferences described in the PDDL3 language [8], and translate them into our formalism. This enables us to investigate the semantics of user preferences in the context of HTNs and provides a clear, unifying framework for augmenting OWL-S processes and preferences.
- We describe our new algorithm, called SCUP (for *Service Composition with User Preferences*). SCUP combines HTN planning with best-first search that uses a heuristic selection mechanism based on ontological reasoning over the input user preferences, state of the world, and the HTNs. In this context, the ontological reasoning provides a way to compute the heuristic cost of a method before decomposing it. Using best-first search, SCUP generates compositions for WSC problems with minimal cost of violations of the user preferences.
- We present an experimental evaluation, demonstrating that our approach is a promising one. On two benchmark problem suites that were used in the latest International Planning Competition (IPC-5) [9], SCUP generated solutions that had substantially better quality than the ones generated by the planning algorithm SGPlan [10, 11], the winner of the IPC-5 in "Planning with Qualitative Preferences" track.

2 Preliminaries

Our formalism is based on the several previous HTN-planning formalisms; in particular, we use similar definitions as for the *Universal-Method Composition Planner (UMCP)* [12–14] and the *HTN-DL* WSC system [4, 15].

We assume the traditional definitions for logical constant and variable symbols, atoms, literals, and ontologies. A *state* is a set of ontological assertions describing the world. A state s is *consistent* if there is an interpretation μ that satisfies all axioms and assertions in s; otherwise we say the state is *inconsistent*.

We assume the existence of finite set of symbols, called *task symbols*. A *task* is an expression of the form $t = (head, \mathsf{Inp}, \mathsf{Out}, \mathsf{Pre}, \mathsf{Eff})$. $head$, the name of this task, is a task symbol. Inp and Out, the set of input and output parameters respectively, are finite sets of variable symbols. Pre and Eff are the preconditions and the effects of the task, respectively, both of which are in the form of conjunctive ontological assertions.

A *task network* is of the form $((n_1 : t_1, n_2 : t_2, \ldots, n_k : t_k), \Delta)$ such that each n_i is a unique label symbol, each t_i is a task, and Δ is a set of *task-network constraints*. Label symbols in a task network are used to differentiate with different occurrences of a particular task in the network. We define two kinds of task-network constraints: *task-ordering* and *state constraints*. A *task-ordering constraint* is an expression of the form $(n \prec n')$, where n and n' are the labels associated with two tasks t and t' in the current task network, respectively. Intuitively, $n \prec n'$ means that the task denoted by the label n must be accomplished before the task denoted by n'.

A *state constraint* is either of the following forms: (n, ϕ), (ϕ, n), or (n, ϕ, n'). Here, ϕ is an arbitrary logical formula, and n, n' are label symbols. Intuitively, state constraints (n, ϕ) and (ϕ, n) mean that the literal ϕ must be true in the state immediately after or before the task labeled with n, respectively. These constraints are used to specify the effects and preconditions of a task labeled by n. A state constraint of the form (n, ϕ, n') means that ϕ is true in every state between n and n'.

If we want a state constraint to hold for every occurrence of the task regardless of the label symbols, we simply use the task name in the constraint, e.g. (ϕ, t). We also use *task variables* in constraint expressions, e.g. $(\phi, ?t)$ and $(?t_1 \prec \ ?t_2)$, in order to represent an existential constraint. Intuitively, a task variable will be bound (only once) to a task that exists in the task network such that the constraint will be satisfied. If no such binding is possible throughout the decomposition process we say that the constraint is violated. In addition, we allow a *task-binding constraint* in the form $(?t = t)$ and $(?t \neq t)$ to restrict the tasks that a task variable can be bound to.

Let s be a state, w be a task network, and x be a task-network constraint in w. Then, we define a *violation-cost function*, χ, as the partial function:

$$\chi(s, w, x) = \begin{cases} c, & \text{if } x \text{ does not hold in } w \text{ given } s, \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where c is either a positive number or it is ∞. The *overall violation cost* of a task network $w = (\tau, \Delta)$ in a state s is

$$\Xi(s, w, \Delta) = \sum_{x \in \Delta} \chi(s, w, x). \tag{2}$$

A *WSC planning operator* is an expression of the form $o = (task, \mathsf{Inp}, \mathsf{Out}, \mathsf{Pre}, \mathsf{Eff})$ where $task$ is a task symbol that specifies the task of this operator, Inp is the set of input parameters, Out is the set of output parameters, Pre is the preconditions of the operator, and Eff is the effects of the operator. Pre and Eff are in the form of conjunctive ontological assertions. An *action* is a ground operator instance which can be executed directly on the Web. A *plan* (or equivalently, a *service composition*) is a sequence of actions. We describe the state change from s to s' when applying an action a by the *state-transition function* γ: that is, $s' = \gamma(s, a)$.

A WSC planning operator o is *applicable* to a task t in a state s, if there is a variable substitution σ such that $\sigma(o)$ is ground, the preconditions of $\sigma(o)$ hold in the state s, o has the same task symbol as in t, and $\sigma(o)$ entails the effects of $\sigma(t)$.

A *task-decomposition method* is a tuple $m = (task, \mathsf{Inp}, \mathsf{Out}, \mathsf{Local}, \mathsf{Pre}, \mathsf{Eff}, \Gamma)$, where $task$, Inp, Out, Pre, and Eff are defined the same as in the above definition of a

planning operator. Γ is a task network. Local is a set of local variables used in P and Γ. Local variables are those variables in m that does not appear in the inputs, but are bound in the preconditions of the method or of a task network. A method m is *applicable* to a task t in a state s, if there is a variable substitution σ such that $\sigma(m)$ is ground, the preconditions of $\sigma(m)$ are satisfied in the state s, m has the same task symbol as in t, and the effects of the $\sigma(m)$ entails the desired effects of $\sigma(t)$.

A *WSC planning domain D* is a triple of (O, M, T_{ont}), where O is the set of operator descriptions, M is the set of methods, and T_{ont} is the task ontology. We assume that the task ontology T_{ont} is an OWL ontology [16] that describes the set of all possible tasks in the domain. A *WSC planning problem* is a triple $P = (s_0, w, D, \Xi)$, where s_0 is the initial state, w is the initial task network, D is the WSC planning domain, and Ξ is the overall violation-cost function. A *solution* for P is a task network $w^* = (\tau, \Delta)$ such that (1) all of the tasks in τ are primitive; (2) there exists a total-ordering of the tasks in τ (i.e., a solution plan $\tau = \langle a_1, a_2, ..., a_k \rangle$) that has a minimum overall cost $\Xi(s_k, w^*, \Delta) < \infty$, where s_k is the final state generated by applying the actions in τ in s_0. Note that a plan that has a violation-cost of ∞ is not a solution even if it somehow generates the final state, when applied in the initial state s_0.

We remark that our formalism above is not intended to depend on a particular WSC system and can be implemented as the input languages of any of the existing HTN-based WSC systems such as [4, 15, 6]. The rationale behind this formalism is that it provides a clear semantics for augmenting OWL-S process models and user preferences in a unifying framework, as we will describe and discuss in the rest of this paper.

3 Modeling User Preferences

Our preference language is largely based on the recent planning language PDDL3 [8] that allows to incorporate user preferences in planning problems. In PDDL3, preferences are described as logical assertions over states and state trajectories by defining *basic preferences* and *temporal preferences*. A *basic preference* (BP) is a logical formula ϕ, or a formula built with logical connectives $\neg$, $\wedge$, $\vee$ from other basic preferences. Let ϕ be a basic preference. Then, a *temporal preference* (TP) is any of the following:[1]

$$\langle TP \rangle := (final\ \langle BP \rangle) \mid (always\ \langle BP \rangle) \mid (sometime\ \langle BP \rangle) \mid (at\_most\_once\ \langle BP \rangle) \mid$$
$$(sometime\_after\ \langle BP \rangle\ \langle BP \rangle) \mid (sometime\_before\ \langle BP \rangle\ \langle BP \rangle)$$

As the above definition demonstrates, we do not allow the nesting of temporal constraints in this paper. We assume that each temporal preference has a unique label; e.g., $p_1 : (final\ \phi)$, where p_1 is the unique label and ϕ is a basic preference.

The satisfaction of temporal constraints is defined with respect to a plan π and its state trajectory $S = \langle s_0, \ldots s_n \rangle$ in an initial state s_0 as follows.

– If ϕ is a ground atom then $\langle \pi, S, s_i \rangle \models \phi$ iff $s_i \models \phi$;

[1] This is the subset of all of the modalities proposed for PDDL3, which is semantically well-developed and used in the International Planning Competition (IPC-5) [9].

- Quantifiers and logical connectives have the same meaning as in first-order logic. For example, $\langle \pi, S, s_i \rangle \models \phi \wedge \phi'$ iff $\langle \pi, S, s_i \rangle \models \phi$ and $\langle \pi, S, s_i \rangle \models \phi'$;
- $\langle \pi, S, s_i \rangle \models always(\phi)$ iff $\forall j : i \leq j \leq n \cdot \langle \pi, S, s_j \rangle \models \phi$;
- $\langle \pi, S, s_i \rangle \models sometime(\phi)$ iff $\exists j : i \leq j \leq n \cdot \langle \pi, S, s_j \rangle \models \phi$;
- $\langle \pi, S, s_i \rangle \models final(\phi)$ iff $\langle \pi, S, s_n \rangle \models \phi$;
- $\langle \pi, S, s_i \rangle \models at\_most\_once(\phi)$ iff (1) $\exists j : i \leq j \cdot \langle \pi, S, s_j \rangle \models \phi$ and $\nexists k : i \leq k \cdot \langle \pi, S, s_k \rangle \models \phi$, or (2) there is no $i \leq j$ such that $\langle \pi, S, s_j \rangle \models \phi$;
- $\langle \pi, S, s_i \rangle \models sometime\_after(\phi, \phi')$ iff (1) $\exists j : i \leq j \cdot \langle \pi, S, s_j \rangle \models \phi$ and $\exists k : j \leq k \cdot \langle \pi, S, s_k \rangle \models \phi'$, or (2) $true$, if $\nexists j : i \leq j \cdot \langle \pi, S, s_j \rangle \models \phi$;
- $\langle \pi, S, s_i \rangle \models sometime\_before(\phi, \phi')$ iff (1) $\exists j : i \leq j \cdot \langle \pi, S, s_j \rangle \models \phi$ and $\exists k : i \leq k \leq j \cdot \langle \pi, S, s_k \rangle \models \phi'$, or (2) $true$, $\nexists j : i \leq j \cdot \langle \pi, S, s_j \rangle \models \phi$;

PDDL3 also allows for representing preferences with universal and existential quantification. As an example, consider the following two different kinds of quantified preferences specified in PDDL3 for a transportation domain:

```
p1A: sometime (forall (?t - truck) at(?t, loc1))
forall (?t - truck) (p1B: (sometime at(?t, loc1)))
```

The preference `p1A` suggests all trucks stay at location $loc1$ sometime in the plan trajectory. If a truck does not satisfy the condition, the preference $p1A$ is violated. The preference `p1B` defines a group of preferences with different truck instantiation: e.g., if there are three trucks in the problem defintion, $p1B$ defines three independent preferences. Violation of one preference will not affect the satisfaction of the others.

4 Augmenting OWL-S with User Preferences

In this section, we describe a translation methodology that takes OWL-S descriptions of Semantic Web services and PDDL3-style preferences as above, and maps them into our planning formalism described in Section 2. The rationale behind this methodology is to represent both the Web services and the preferences in a single unifying framework, in which a planning algorithm could work to generate compositions of services while satisfying the user preferences as much as possible.

From OWL-S to HTN-based Constraints. Our translation method for encoding OWL-S in our HTN-based constraint language is very similar to the technique described in [4] for encoding OWL-S in the input language of the SHOP2 planner [17].

Suppose K is a collection of OWL-S process models. First of all, for every process C in K, we create a task of the form $(C\ \boldsymbol{v}\ \boldsymbol{u}\ \boldsymbol{p}\ \boldsymbol{e})$ where $\boldsymbol{v}$ and $\boldsymbol{u}$ are the list of input and output parameters defined in D_C, $\boldsymbol{p}$ and $\boldsymbol{e}$ are the preconditions and effects of D_C, and a corresponding label n_C for that task.

We translate each atomic process in K into a corresponding planning operator in the same way as in [4]. The translation of a composite process C in K with a **sequence** control construct is as follows. Let D_C be the OWL-S definition of the process C.

Next, we define an HTN method m in our constraint language whose head is C and whose inputs, outputs, preconditions, and effects are $\boldsymbol{v}$, $\boldsymbol{u}$, $\boldsymbol{p}$ and $\boldsymbol{e}$, respectively. The task network specified by the method m is translated from the process C as follows:

1. For each precondition p defined in D_C, we create a state-constraint of the form (p, n_C) where n_C is the label of the task corresponding to the process C.
2. For the sequence $C_1, C_2, \ldots, C_k$ of composite process defined in the **sequence** control construct of C, we do the following.
   - We define a task $t_i = (C_i\ \boldsymbol{v}_i\ \boldsymbol{u}_i\ \boldsymbol{p}_i\ \boldsymbol{e}_i)$ as above, and a corresponding task label n_i. The task network in the method m contains the tasks defined as $((n_i : t_i))_{i=1}^{k}$.
   - For each $1 \leq i < k$, we define an ordering constraint $(n_i \prec n_{i+1})$ where n_i and n_{i+1} are task labels for the tasks that correspond to the processes C_i and C_{i+1}.

Both sets of constraints above go into the constraint definition Δ of the task network associated with the method m.

The translation of a process C with an **if-then-else** control construct is as follows. Let P_{if} be the conditions for the **if** clause as defined in D_C. We define two HTN methods m_{if} and m_{else} to encode C such that the heads of both methods are the same task $(C\boldsymbol{v})$. Then, we do the following translation for the constraint part of those methods:

- For m_{if}:
  1. For each precondition p either defined in D_C or in P_{if}, we define a state-constraint (p, n_C) where n_C is the label of the task corresponding to C.
  2. For the process C_{then} defined for the **then** construct in D_C, we define a task $t = (C_{then}\ \boldsymbol{v}_{then}\ \boldsymbol{u}_{then}\ \boldsymbol{p}_{then}\ \boldsymbol{e}_{then})$ as above, and a corresponding label n. The task list of the task network associated with the method m_{if} is $((n : t))$.
- For m_{else}:
  1. For each precondition p defined in D_C, we define a state-constraint of the form (p, n_C) where n_C is the label of the task corresponding to the process C.
  2. For each precondition p defined in P_{if}, we define a state-constraint of the form $(\neg p, n_C)$ where n_C is the label of the task corresponding to the process C.
  3. For the process C_{else} defined for the **else** construct in D_C, we define a task $t = (C_{else}\ \boldsymbol{v}_{else}\ \boldsymbol{u}_{else}\ \boldsymbol{p}_{else}\ \boldsymbol{e}_{else})$ as above, and a corresponding label n. The task list of the task network associated with the method m_{else} is $((n : t))$.

The translations for **repeat-while**, **repeat-until**, **choice**, and **unordered** constructs that may appear in the processes in K are similar to the ones above.

For every HTN task-network constraint generated with our translation, we assign a violation-cost of ∞, independent of the state and task network that the constraints might be evaluated in. By doing so, we ensure that the translation generates a bijection between the OWL-S process models (where such constraints must be satisfied with absolute certainty) and the translated HTN constructs.

Proposition 1. *The translation from OWL-S service process models into HTN constraints is correct, i.e. for each OWL-S process construct as described above, there is a corresponding set of HTN constructs that specify the same process-model semantics.*

Translating Preferences into HTN Constraints. We will now describe how to translate a preference ψ to one compound task-network constraint $\Gamma(\psi)$. With this translation, whenever we augment a task network w with $\Gamma(\psi)$, we have a direct method to evaluate if a state trajectory (i.e., a plan) satisfies or violates the preference.

In our translation, we use two special task symbols t_{start} and t_{end} as the start and end tasks that will be added to every initial task network w with labels n_{start} and n_{end}. We also add the ordering constraint $n_{start} \prec n$ and $n \prec n_{end}$ for every $n : t \in w$.

We use the special symbol $\top$ to denote a trivially satisfiable HTN constraint and $\bot$ to denote an unsatisfiable HTN constraint with the usual semantics that $\neg\top = \bot$, $\neg\bot = \top$, $\top \wedge \phi = \psi$, $\top \vee \psi = \top$, $\bot \wedge \psi = \bot$ and $\bot \vee \psi = \psi$. If $\Gamma(\psi) = \top$, then every plan satisfies ψ and if $\Gamma(\psi) = \bot$ there is no plan that can satisfy ψ.

Now we describe how we construct $\Gamma(\psi)$ by analyzing the temporal preferences separately:

- $\Gamma(\psi) \equiv (n_{start}, \phi, n_{end})$, if $\psi \equiv always(\phi)$, where ϕ is a basic preference.
- $\Gamma(\psi) \equiv (?t, \phi)$, if $\psi \equiv sometime(\phi)$, where ϕ is a basic preference.
- $\Gamma(\psi) \equiv (\phi, n_{end})$, if $\psi \equiv final(\phi)$, where ϕ is a basic preference.
- $\Gamma(\psi) = (?t_1, \phi) \wedge (?t_2, \phi') \wedge (?t_1 \prec ?t_2)$, if $\psi \equiv sometime\_after(\phi, \phi')$.
- $\Gamma(\psi) = (?t_1, \phi) \wedge (?t_2, \phi') \wedge (?t_2 \prec ?t_1)$, if $\psi \equiv sometime\_before(\phi, \phi')$.
- $\Gamma(\psi) = (\phi, ?t)$ and for every task t, $(\phi, t) \implies ?t = t$, if $\psi \equiv at\_most\_once(\phi)$.

We translate the universally- and existentially-quantified preferences as follows:

- If $\psi \equiv \forall(?x) \cdot \phi$, where $?x$ is a variable symbol, then we apply the following rules:
  1. If ϕ is a basic preference, then $\Gamma(\psi) \equiv \Gamma(\phi(?x = x_1) \wedge \phi(?x = x_2) \wedge \ldots \wedge \phi(?x = x_n))$, for all possible instantiations x_i of $?x$, $i = 1, \ldots, n$.
  2. If ϕ is a temporal preference, then $\Gamma(\psi) \equiv \{\Gamma(\phi(?x = x_1)), \Gamma(\phi(?x = x_2)), \ldots, \Gamma(\phi(?x = x_n))\}$, for all possible instantiations x_i of $?x$, $i = 1, \ldots, n$.
- We have $\psi \equiv \exists(?x)(\phi)$ as $\Gamma(\psi) \equiv \phi$ since existential quantification has the same logical meaning for both basic and temporal preferences.

Proposition 2. *The translation from preferences into HTN constraints is correct: if $\pi = \langle a_1, a_2, \ldots, a_k \rangle$ is a solution plan to a WSC planning problem (s_0, w, D, Ξ) with a violation cost of v, where the user preferences ψ are translated into HTN constraints $\Gamma(\psi)$ in the task network w, then $v = \Xi(s_k, w^*, \Gamma(\psi))$, where $w^* = (\pi, \Delta)$, s_k is the final state generated by applying π in s_0 and $\Gamma(\psi) \subseteq \Delta$.*

5 SCUP: Service Composition with User Preferences

We are now ready to describe our new WSC planning algorithm, SCUP, for generating service compositions while satisfying user preferences as much as possible. Figure 1 shows the abstract description of the SCUP algorithm. SCUP takes five inputs s_0, w_0, D, ψ, and Ξ. s_0 is the initial state and w_0 is the initial task network. D is the

Procedure SCUP(s_0, w_0, D, ψ, Ξ)

1. $\pi \leftarrow \emptyset$
2. $w = (\tau, \Delta) \leftarrow$ Preprocess(w_0, D, R)
3. $OPEN \leftarrow \{(s_0, w, \pi, \Xi(s_0, w, \Delta))\}$
4. while $OPEN \neq \emptyset$ do
5. select the first node $x = (s, w, \pi, v)$ from $OPEN$ and remove it
6. if π is a solution then **return** π
7. $Succ \leftarrow$ Decompose(s, w, π, v, D)
8. $OPEN \leftarrow OPEN \cup$
   $\{(s', w', \pi', \Xi(s', w', \Delta')) \mid (s', w' = (\tau', \Delta'), \pi') \in Succ\}$
9. sort $OPEN$ in the ascending order based on the cost function Ξ
10. **return** $failure$

Fig. 1. An abstract description of the SCUP algorithm.

domain knowledge including the task-ontology definitions, HTN methods, and planning operators. ψ is the set of user preferences and Ξ is the overall violation cost function.

The algorithm first generates the task network w, where the tasks in $w_0 = (\tau, \Delta)$ are rearranged based on the ordering constraints in w_0 and the ordering constraints in the translation $\Gamma(\psi)$ of the input preferences in ψ. The Preprocess subroutine is responsible for this operation (Line 2 of the pseudocode in Figure 1). Preprocess first scans all preferences and see if there are `final`, `sometime_before`, `sometime_after`, and `sometime` preferences. If there are no such preferences in ψ, the algorithm simply returns the original task network w_0 with $\Delta \cup \Gamma(\psi)$. Otherwise, Preprocess checks all of the preferences of the above types and verify if additional tasks will be added in w_0 in order to guarantee the satisfaction of these preferences in ψ as much as possible.

The rationale behind the preprocessing step is as follows:

- **Preprocessing (`sometime` ϕ).** If no task in τ can achieve the effect ϕ, there might be still other tasks in the input task ontology, when/if decomposed into a primitive HTN successfully, can achieve ϕ. Thus, Preprocess searches for such an additional task t whose effect entails ϕ. If there is such a task in the input task ontology, Preprocess adds t in τ and the state constraint (t, ϕ) into Δ. The state-constraint (t, ϕ) ensures that the condition ϕ is supposed to be achieved after the decomposition of t is done. As a result, if the task t is accomplished successfully during planning, then the preference ϕ will be satisfied as well.
- **Preprocessing (`sometime_before` ϕ ϕ').** In this case, Preprocess generates two sets of tasks T_1 and T_2 that can accomplish the effect ϕ and ϕ', respectively. For each task t in T_1 with a label n, the subroutine chooses a task t' in T_2 with a label n', and adds the task-network constraints $(n \prec n')$, (n, ϕ), and (n', ϕ') to Δ.

The preprocessing of `final` and `sometime_after` preferences are are similar to `sometime` and `sometime_before` as described above.

After adding all task-ordering constraints into the task network w_0 based on the preferences in ψ, the set of ordering and state constraints in w_0 may not be satisfiable. For example, in a transportation-services domain as in [6], we may have both

$(n_1 : load(truck1, package1) \prec n_2 : drive(truck1))$ and $(n_2 : drive(truck1) \prec n_1 : load(truck1, package1))$ in ψ. As another example, we may also have a cyclic constraint such as $(n_1 : drive(truck1) \prec n_2 : load(truck1, package1))$, $(n_2 : load(truck1, package1) \prec n_3 : deliver(package1))$ and $(n_3 : deliver(package1) \prec n_1 : drive(truck1))$ in w_0. There is no solution plan which can satisfy these HTN constraints during the planning process. Thus, Preprocess, in such cases, greedily removes the constraint with less weight from w_0 in order to satisfy more important preferences (i.e., preferences with smaller costs).

A couple of remarks on Preprocess are in order. First, it is important to note that the additional HTN constraints that Preprocess adds to Δ is due to the translation Γ of those preferences based on the additional tasks from the task ontology. Second, we only work with `final`, `sometime_before`, `sometime_after`, and `sometime` preferences in the preprocessing phase. The reason is that we can improve the cost value by adding additional tasks for `sometime` and `final`, and rearrange task orderings for `sometime_before`, `sometime_after`, and `sometime`. As for other preferences like `always` and `at_most_once`, we can only check their satisfaction in decomposition and choose the best tuple in $OPEN$ list by using best first search.

After rearranging the task network $w = (\tau, \Delta)$, the algorithm computes the overall violation-cost value $v = \Xi(s_0, w, \Delta)$ and puts the tuple (s_0, w, π, v) in the $OPEN$ list (see Line 3), where π is the current partial plan. As we mentioned above, SCUP is an HTN task-decomposition planner that is based on a best-first search procedure. At each iteration of the loop, SCUP first takes the first tuple $(s, w, \pi, v) \in OPEN$ and removes it. If π is a solution composition (i.e., plan), then SCUP returns π. Otherwise, the algorithm decomposes the task-network w according to the task-decomposition methods in D, generates the set $Succ$ of successor state and task-network pairs, computes the violation costs for those new pairs in $Succ$, and puts them back in $OPEN$ (see Line 9). The tuples in $OPEN$ are then sorted in the ascending order of their violation costs. This process continues until SCUP finds a solution or fails to generate a plan.

In Line 8 of Figure 1, the Decompose subroutine first nondeterministically chooses a task t in w with no predecessors and finds the set A of all applicable operators and methods for task t. If it cannot find any applicable operators or methods, it simply returns failure. Otherwise, for each item u in A, Decompose does the following:

- If u is an operator, Decompose first generates the action a by applying input bindings to u. Then, it computes the next state $s' = \gamma(s, a)$, where s is the current state, and the successor task-network w' by removing t from w. It appends a to the partial plan π.
- If u is a method, the subtasks w'' will replace task t in w. The effects of each task in w'' is then used to check if any preference is satisfied or violated in the current state, and thus, to update the overall cost. This helps us foresee the violation costs without further decomposition, and decide whether we should continue on this task network. Decompose adds in $Succ$ the tuple with the same state s, the updated task nework w' by replacing t with w'' in w, and the same partial plan π.

After the loop over all applicable operators and methods, Decompose checks and removes duplicate tuples in $Succ$ that are those which have the same task network and partial plan, in order not to repeat the same decompositions during planning.

An example of how Decompose uses the task effects to compute the heuristic violation costs is in order. First, recall that the preprocessing phase generates the post-condition state-constraint (t, ϕ) for each task t that is inserted into the task network due to a preference. Once the planner completely decomposes t into a primitive task network, Decompose checks if the post-condition ϕ is satisfied in the current state.

For example, suppose we have a task $SendPackage(package1, location3)$ with an post-condition $delivered(truck3, package1)$ due to a preference (`p1 : sometime` $delivered(truck3, package1)$). The following is an HTN method for $SendPackage$:

method `SendPackage`
Inputs: (?pac ?loc)
Local: (?truck)
Preconditions: (available(?truck))
Subtasks: (load(?truck, ?package), deliver(?truck, ?pac, ?loc))
Effects: (delievered(?truck, ?pac))

Note that $truck3$ is not the input to $SendPackage$, and the method uses a variable $?truck$ to match any available trucks near $package1$. Suppose the planner finds three trucks $truck1$, $truck2$, $truck3$ in the current state. Then, it will create three tuples in $Succ$ each possible truck. However, since $SendPackage$ has the post-condition $delivered(truck3, package1)$, only the tuple with the truck variable binding $truck3$ will satisfy the preference $p1$ and the others will violate it, increasing their cost values. Thus, when the tuples in $Succ$ are merged into the $OPEN$ list and sorted based on the costs of its tuples, SCUP will always consider the tuple that satisfies the preference $p1$.

6 Implementation and Preliminary Experiments

We have implemented a prototype of the SCUP algorithm. Our current prototype is built on our previous system HTN-DL [15]. HTN-DL does not directly implement our planning language as we described in Section 2. Thus, we assumed that the translation from OWL-S to HTN-DL has been done automatically as described in [15] and we implemented the translation of user preferences from PDDL3 into HTN-DL's input language. Furthermore, we used the interface functionality between HTN-DL and Pellet, an OWL DL reasoner [18], that we used in our SCUP prototype for knowledge inference and ontological reasoning.

During planning, the size of SCUP's $OPEN$ list may get very large and this may induce serious performance drawbacks for the SCUP algorithm. For that reason, we implemented an additional input parameter k that bounds the size of the $OPEN$, which, as a trade-off, may affect the solution plan quality. If k is too small, a tuple in $OPEN$ with prospective solution may be discarded in the first few decompositions. This happens when the local optimal search node is not the global optimal search node. In our experiments described below, we used $k = 20$ as the size of the $OPEN$ list and this was sufficient to generate optimal (or in some cases, near-optimal) solutions.

We have conducted experiments with two benchmark planning domains used in "Planning with Qualitative Preferences" track of the most recent International Planning Competition (IPC-5) (`http://zeus.ing.unibs.it/ipc-5/`): Trucks and Rovers. In the Trucks domain, the goal is to move packages between locations by using trucks. There are only four operators $load$, $drive$, $unload$, and $deliver$. Each truck may have multiple truck areas so as to carry packages, but there are constraints and penalties on the ordering of loading packages and the input user preferences model choices over the trucks based on those constraints. Generally, loading multiple packages will result in preference violations, and on the other hand, delivering only a single package at a time may delay the delivering deadlines of other packages.

In the Rovers domain, the goal is to navigate the rovers on a planet surface, collect scientific data such as rock and soil samples, and send them back to a lander. These tasks need to be achieved by considering the input spatial constraints and temporal preferences on the operation of a rover. Each rover has only a limited storage capacity for the collected samples and it is only capable of sampling either soil, rock, image, or some combination of them specified in the problem descriptions. A rover can only travel between certain waypoints only when the path from source to destination is visible.

We used the exact planning domain and problem descriptions that were used in IPC-5. For each planning domain, we have 20 problems with increasing number of instances, goals and preferences. In our experiments, we compared the overall violation cost values of the solutions generated by SCUP and SGPlan [10, 11]. SGPlan is an AI planning algorithm that participated in the "Planning with Qualitative Preferences" track of the IPC-5, and won the 1^{st} place. We have not run the SGPlan ourselves for our experiments; instead, we used the published results for it from the IPC-5 [9]. Note that this does not affect our results since we are comparing SCUP with SGPlan on the costs of their solutions and neither of the planners had any memory issues in our experiments.

Figure 2 shows our results in the Trucks domain. SCUP has generated solutions that have cost values that are substantially less than SGPlan in all 20 problems, where SCUP satisfied all preferences in 8 of the experimental problems. The average cost value for SCUP is 2.00, less than one-third of SGPlan's average 7.45. Most violations in SCUP solutions result from the conflicts between preferences; e.g., delivering multiple packages on a particular truck is necessary to satisfy delivery deadline preferences that had larger costs, and this unavoidably causes violations of using multiple truck areas.

Figures 3 and 4 illustrate how SCUP and SGPlan performed in the Rovers domain. SCUP has outperfermed SGPlan in 19 out of 20 problems. In these experiments, we found that as the size of the problems is increased, SCUP discarded possible solution tuples due to the $OPEN$ list size limitation. However, generally SCUP has higher quality solutions with average cost value 888.645, compared to SGPlan's 1608.378.

7 Related Work

There has been many advances in WSC planning in the recent years [2, 4, 6, 19–21]. Probably the first work in this research area is the one described in [2], where the states of the world and the world-altering actions are modeled as Golog programs, and the information-providing services are modeled as external functions calls made within

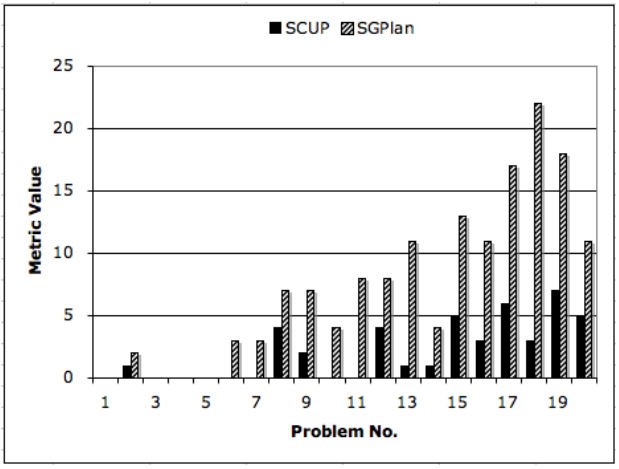

Fig. 2. The violation costs of the solutions generated by SCUP and SGPlan on the Trucks problems.

those programs. The goal is stated as a Prolog-like query and the answer to that query is a sequence of world-altering actions that achieves the goal, when executed in the initial state of the world. During the composition process, however, it is assumed that no world-altering services are executed. Instead, their effects are simulated in order to keep track of the state transitions that will occur when they are actually executed.

In [4], the WSC procedure is based on the relationship between the OWL-S process ontology [7] used for describing Web services and *Hierarchical Task Networks* as in HTN Planning [17]. OWL-S processes are translated into tasks to be achieved by the SHOP2 planner [17], and SHOP2 generates a collection of atomic process instances that achieves the desired functionality. [6] extended this work to cope better with the fact that information-providing Web services may not return the needed information immediately, or at all. The ENQUIRER algorithm presented there does not cease the search process while waiting answers to some of its queries, but keeps searching for alternative compositions that do not depend on answering those specific queries.

In all works above, search for desirable solutions have been incorporated into the service-composition process as *hard constraints*; i.e., constraints that must be satisfied by all of the solutions. Recently, several different approaches have been developed for planning with preferences; i.e., *soft constraints* that are *preferably* but *not necessarily* satisfied by a plan. There are various different approaches for integrating user preferences in the planning process. Examples include [22, 23, 10, 24].

In our experimental study here, we considered one of those state-of-the-art planners, SGPlan [10, 11], that won the recent International Planning Competition in the "Planning with Qualitative Preferences" track. SGPlan is a planning algorithm that uses a divide-and-conquer approach: the planner serializes a large planning problem into subproblems with subgoals, solves the sub-problems, merges the solutions to those subproblems, and tries to remove conflicts between them. It has been demonstrated in ex-

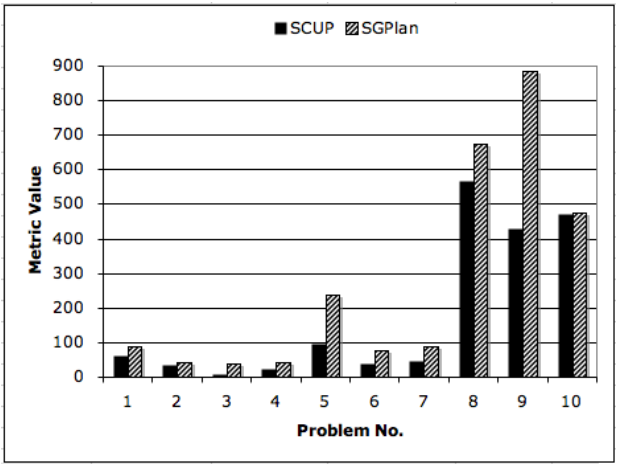

Fig. 3. The violation costs of the solutions generated by SCUP and SGPlan on the smaller Rover problems.

perimental studies that this approach largely reduces the search space compared to the state-space of the original problem.

In another work that is very related to our paper, Baier et al. proposed the heuristic-search planner HPlan-P in [24]. They used a best-first search algorithm with a goal distance function to find the first solution, and tried to improve solution by using other heuristics. Unfortunately, we were not able to get the HPlan-P planner for experimentation due to its distribution-licensing problems at the time we personally communicated with the authors of that planner, and since SGPlan significantly outperformed HPlan-P in IPC-5, we have not included its results here.

8 Conclusions and Future Work

In many interesting Web Service Composition (WSC) problems, the goal is to generate desirable solutions with respect to some user preferences that are not necessarily absolute objectives or constraints, but a composition algorithm needs to try to satisfy them as much as possible. In this paper, we have described a novel approach for incorporating user preferences in planning for Web Service Composition. We have first described a way to take OWL-S service descriptions and PDDL3-style preferences, and translate them into a planning language for Hierarchical Task Networks (HTNs). Based on this translation, we have then described an HTN planning algorithm, called SCUP, that combines HTN planning and best-first search for WSC planning with preferences.

Our preliminary experiments demonstrated that SCUP is a promising approach: our prototype implementation of SCUP was able to generate solutions that satisfy more user preferences and those preferences that have more value (i.e., less cost) than the state-of-the-art planner SGPlan, the winner of the latest International Planning Compe-

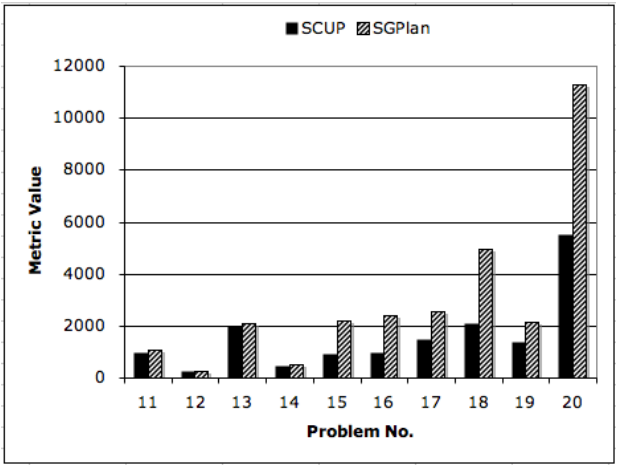

Fig. 4. The violation costs of the solutions generated by SCUP and SGPlan on the larger Rover problems.

tition in the "Planning with Qualitative Preferences" track. In the near future, we are planning to conduct extensive theoretical and experimental evaluation of our approach. In a particular future study, we will investigate the relationships between the amount of information needs to be gathered during the composition process, and the time it takes for SCUP to optimal and near optimal solutions. Furthermore, SCUP currently handles different type of preferences separately, and this may result in local optimum instead of global optimum in certain domains. We will investigate this hypothesis both theoretically and experimentally in the near future.

References

1. McDermott, D.: Estimated-regression planning for interactions with web services. In: AIPS. (2002) 204–211
2. McIlraith, S., Son, T.: Adapting Golog for composition of semantic web services. In: KR-2002, Toulouse, France (apr 2002)
3. Martinez, E., Lespérance, Y.: Web service composition as a planning task: Experiments using knowledge-based planning. In: Proceedings of the ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services. (June 2004) 62–69
4. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Journal of Web Semantics **1**(4) (2004) 377–396
5. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: International Semantic Web Conference. (2004) 380–394
6. Kuter, U., Sirin, E., Nau, D., Parsia, B., Hendler, J.: Information gathering during planning for web services composition. In: ISWC-2004. (2004)
7. W3C: Owl-based web service ontology `www.daml.org/services/owl-s/`.
8. Gerevini, A., Long, D.: Plan constraints and preferences in pddl3 (2006)

9. Gerevini, A., Long, D.: The fifth international planning competition (2006) http://ipc5.ing.unibs.it/.
10. Hsu, C.W., Wah, B.W., Huang, R., Chen, Y.X.: Handling soft constraints and preferences in sgplan. In: ICAPS Workshop on Preferences and Soft Constraints in Planning. (June 2006)
11. Hsu, C.W., Wah, B.W., Huang, R., Chen, Y.X.: Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In: IJCAI. (2007) 1924–1929
12. Erol, K., Hendler, J., Nau, D.S.: UMCP: A sound and complete procedure for hierarchical task-network planning. In: Proceedings of the International Conference on AI Planning Systems (AIPS). (June 1994) 249–254
13. Erol, K., Hendler, J., Nau, D.S., Tsuneto, R.: A critical look at critics in HTN planning. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). (1995)
14. Erol, K., Hendler, J., Nau, D.S.: Complexity results for hierarchical task-network planning. Annals of Mathematics and Artificial Intelligence **18** (1996) 69–93
15. Sirin, E.: Combining Description Logic Reasoning with AI Planning for Composition of Web Services. PhD thesis, Department of Computer Science, University of Maryland (2006)
16. W3C: Web ontology language (owl) (2004) www.w3c.org/2004/OWL/.
17. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. JAIR **20** (December 2003) 379–404
18. Sirin, E.: Pellet: The open source owl dl reasoner (2006) http://pellet.owldl.com/.
19. Martinez, E., Lespérance, Y.: Web service composition as a planning task: Experiments using knowledge-based planning. In: ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services. (June 2004)
20. Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., Traverso, P.: Planning and monitoring web service composition. In: AIMSA. (2004)
21. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: ISWC. (2004)
22. Son, T., Pontelli, E.: Planning with preferences using logic programming. In: Proc. LPNMR 2004. (2004)
23. Bienvenu, M., Fritz, C., McIlraith, S.: Planning with qualitative temporal preferences. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06). (2006)
24. Baier, J.A., Bacchus, F., McIlraith, S.A.: A heuristic search approach to planning with temporally extended preferences. In: IJCAI. (2007)